US-PAT-NO: 6578068

DOCUMENT-IDENTIFIER: US 6578068 B1

TITLE: Load balancer in environment services patterns

---------- KWIC ---------

Detailed Description Text - DETX (4):
In general, OOP components are reusable **software** modules which present an interface that conforms to an object model and which are accessed at **run**-time is through a component integration architecture. A component integration architecture is a set of architecture mechanisms which allow software modules in different process spaces to utilize each others capabilities or functions. This is generally done by assuming a common component object model on which to build the architecture. It is worthwhile to differentiate between an object and a class of objects at this point. An object is a single instance of the class of objects, which is often just called a class. A class of objects can be viewed as a blueprint, from which many objects can be formed.

Detailed Description Text - DETX (14):
Frameworks also represent a change in the way programmers think about the interaction between the code they write and code written by others. In the early days of procedural programming, the programmer called **libraries** provided by the operating system to perform certain tasks, but basically the **program** **executed** down the page from **start** to finish, and the programmer was solely

responsible for the flow of control. This was appropriate for printing out
paychecks, calculating a mathematical table, or solving other problems with a
program that executed in just one way.

Detailed Description Text - DETX (47):
The **execution** architecture is a unified collection of **run**-time technology
services, control structures, and supporting infrastructure upon which
application **software runs**.

Detailed Description Text - DETX (55):
It includes components such as: Job scheduler Software distribution Error
monitor **Data** backup and restore Help desk Security administration
High-Availability Hardware management Performance monitors **Startup**/shutdown
procedures Report management tool Disaster Recovery Network Monitoring Tools
Cross Platform Management Tools

Detailed Description Text - DETX (85):
An application style defines a unique class of processing type, which is
used by applications, and thus end-users. Delivery Vehicle Reference set of
Application Styles include batch, on-line transaction processing,
collaboration, data warehouse, knowledge management and integration.

Detailed Description Text - DETX (139):
Business Imperatives 802 B1. The application will be used only by an
internal user community. **Software** distribution is a concern for traditional
client server computing environments due to the fact that **executable** and data
files need to reside on the client hard drive. Distribution to a user

community outside of the client's organization is even more difficult to implement and manage and will probably be limited to a few key business partners. B2. The application requires an advanced, dynamic, and integrated user interface for expert users. State of the art 4GL and 3GL development languages will support advanced user interfaces which require a significant degree of context management between fields and windows. Web-based user interfaces do not support such interfaces well yet. B3. Session performance is critical to the application or sub-second response times are required for successful use. Client server applications can provide response times necessary to support transaction intensive mission critical systems. Application logic and business data can be distributed between the client and server for optimal efficiency. Web-based interfaces still have an inherent overhead due to the connectionless communication and constant downloading of data, formatting information and applet code. B4. The application needs to support off-line mobile users. Mobile computing is becoming more prevalent in the work place, therefore, connectivity to a server can not be assumed for all user classes. A client server architecture allows for the distribution of application logic and/or data between the server and client. Replication of data and logic is usually necessary for applications that are run on portable computers.

Detailed Description Text - DETX (297):

Plug-in--a term coined by Netscape, a plug-in is a **software program** that is specifically written to be **executed** within a browser for the purpose of providing additional functionality that is not natively

supported by the browser, such as viewing and playing unique data or media types. Typically, to use a plug-in, a user is required to download and install the Plug-in on his/her client machine. Once the Plug-in is installed it is integrated into the Web browser. The next time a browser opens a Web page that requires that Plug-in to view a specific data format, the browser initiates the execution of the Plug-in. Until recently Plug-ins were only accessible from the Netscape browser. Now, other browsers such as Microsoft's Internet Explorer are beginning to support Plug-in technology as well. Also, Plug-ins written for one browser will generally need to be modified to work with other browsers. Plug-ins are also operating system dependent. Therefore, separate versions of a Plug-in may be required to support Windows, Macintosh, and Unix platforms.

Detailed Description Text - DETX (298):

Helper Application/Viewer--is a software program that is launched from a browser for the purpose of providing additional functionality to the browser. The key differences between a helper application or sometimes called a viewer and a plug-in are: How the program is integrated with the Web browser--unlike a plug-in, a helper application is not integrated with the Web Browser, although it is launched from a Web browser. A helper application generally runs in its own window, contrary to a plug-in which is generally integrated into a Web page. How the program is installed--like a plug-in, the user installs the helper application. However, because the helper application is not integrated with the browser, the user tends to do more work during installation specifying additional information needed by the browser to launch the

helper application.
How the program is initiated--the user tends to initiate the launching of the
helper application, unlike a plug-in where the browser does the initiation.
From where the **program is executed**--the same helper application can be **executed**
from a variety of browsers without any updates to the **program,** unlike a plug-in
which generally needs to be updated for specific browsers. However, helper
applications are still operating system dependent.

Detailed Description Text - DETX (299):
Java applet--a **program** written in Java that **runs** within or is launched from
the client's browser. This **program** is loaded into the client device's memory
at **runtime** and then unloaded when the application shuts down. A Java applet
can be as simple as a cool animated object on an HTML page, or can be as
complex as a complete windows application running within the browser.

Detailed Description Text - DETX (300):
ActiveX control--is also a **program** that can be **run** within a browser, from an
application independent of a browser, or on its own. ActiveX controls are
developed using Microsoft standards that define how re-usable software
components should be built. Within the context of a browser, ActiveX controls
add functionality to Web pages. These controls can be written to add new
features like dynamic charts, animation or audio.

Detailed Description Text - DETX (343):
Direct Manipulation Services enable applications to provide a direct
manipulation interface (often called "drag & drop"). A direct manipulation
interface allows users to manage multiple "application

objects" by manipulating
visual representations of those objects. For example, a user may sell stock by
dragging "stock" icons out of a "portfolio" icon and onto a "trading floor"
icon. Direct Manipulation Services can be further divided as follows: Display:
These services enable applications to represent application objects as icons
and control the display characteristics (color, location, etc.) of these icons.
Input/Validation: These services enable applications to invoke validation or
processing logic when an end user "acts on" an application object. "Acting on"
an object may include single clicking, double clicking, dragging, or sizing.

Detailed Description Text - DETX (409):
Storage Services manage data physical storage. These services provide a
mechanism for saving information so that data will live beyond **program**
execution. Data is often stored in relational format (an RDBMS) but may also
be stored in an object-oriented format (OODBMS) or other formats such as IMS,
VSAM, etc.

Detailed Description Text - DETX (594):
When trying to decide whether to use MOM technology, keep the following
characteristics of this type of middleware in mind: MOMs are high speed,
generally connectionless and are usually deployed for executing applications
with a nonblocking sender MOM solutions are especially useful for
inter-application communication and are increasingly popular for
inter-enterprise work MOMs support end-to-**end business applications** and process
inter-operability MOMs are designed for heavily **used** production applications
and are generally capable of high throughput rates and fast

transfer times.
Data is usually forwarded immediately, although it is possible to store it for
later processing

Detailed Description Text - DETX (600):
Streaming is an emerging technology. While some multimedia products use
proprietary streaming mechanisms, other products incorporate standards. The
following are examples of emerging standards for streaming protocols. Data
streams are delivered using several protocols that are layered to assemble the
necessary functionality. Real-time Streaming Protocol (RTSP)--RTSP is a draft
Internet protocol for establishing and controlling on-demand delivery of
real-time data. For example, clients can use RTSP to request specific media
from a media server, to issue commands such as play, record and pause, and to
control media delivery speed. Since RTSP simply controls media delivery, it is
layered on top of other protocols, such as the following. Real-Time Transport
Protocol (RTP)--Actual delivery of streaming data occurs through real-time
protocols such as RTP. RTP provides end-to-**end data delivery for applications**
transmitting real-**time** data over multicast or unicast network services. RTP
conveys encoding, timing, and sequencing information to allow receivers to
properly reconstruct the media stream. RTP is independent of the underlying
transport service, but it is typically used with UDP. It may also be used with
Multicast UDP, TCP/IP, or IP Multicast. Real-Time Control Protocol (RTCP)--RTP
is augmented by the Real-Time Control Protocol. RTCP allows nodes to identify
stream participants and communicate about the quality of data delivery.

Detailed Description Text - DETX (687):
Design techniques for integration with existing systems can be grouped into
two broad categories: Front **end access**--discussed as part of Terminal Emulation
Back **end access**--tend to be **used** when existing data stores have information
that is needed in the client/server environment but accessing the information
through existing screens or functions is not feasible. Legacy Integration
messaging services typically include remote data access through gateways. A
database gateway provides an interface between the client/server environment
and the legacy system. The gateway provides an ability to access and
manipulate the data in the legacy system.

Detailed Description Text - DETX (697):
SafePassage is a full-strength, encrypting Web proxy. It is designed to
supplement the security of browsers whose authentication and encryption
capabilities have been weakened to comply with United States export
regulations. For these types of browsers, SafePassage will provide client
authentication certificates and full-strength encryption (128 bit).

Detailed Description Text - DETX (723):
Authentication can occur through various means: Basic Authentication--requires that the Web client supply a user name and password
before servicing a request. Basic Authentication does not encrypt the password
in any way, and thus the password travels in the clear over the network where
it could be detected with a network sniffer program or device. Basic
authentication is not secure enough for banking applications or anywhere where
there may be a financial incentive for someone to steal someone's account

information. Basic authentication is however the easiest mechanism to setup
and administer and requires no special software at the Web client. ID/Password
Encryption--offers a somewhat higher level of security by requiring that the
user name and password be encrypted during transit. The user name and password
are transmitted as a scrambled message as part of each request because there is
no persistent connection open between the Web client and the Web server.
Digital **Certificates or Signatures--encrypted** digital keys that are issued by a
third party "trusted" organization (i.e. Verisign); used to **verify** user's
authenticity. Hardware tokens--small physical devices that may generate a
one-time password or that may be inserted into a card reader for authentication
purposes. Virtual tokens--typically a file on a floppy or hard drive used for
authentication (e.g. Lotus Notes ID file). Biometric identification--the
analysis of biological characteristics to verify individuals identify (e.g.,
fingerprints, voice recognition, retinal scans).

Detailed Description Text - DETX (738):
Provides the underlying protocols responsible for transmitting and securing
data communications. Transport Services are responsible for establishing,
maintaining and terminating **end-to-end communications** between users and
processes. Connection management provides transfer services that ensure the
delivery of data from sender to receiver, which support the transferring of
messages from a process running on one machine to a process running on another
machine. In addition, connection management provides services that initiate a
connection, gracefully terminate a connection, and handle abrupt termination.
These services take place for application before and after

the data is
formatted for transport over the network.

Detailed Description Text - DETX (778):
The following are examples of standards that implement Network Address
Allocation and allow a network node to ask a central resource for the node_s
network address (e.g., IP address): DHCP (Dynamic **Host** Configuration Protocol)
BootP (Bootstrap Protocol)

Detailed Description Text - DETX (781):
The following list provides a description of various Quality of Service
parameters. connection establishment delay--time between the connection
request and a confirm being received by the requester connection establishment
failure probability--chance that the connection will not be established within
the maximum establishment delay throughput--bits per second (bps) of
transmitted **data** transit delay--time elapsed between when sender transfers
packet and recipient receives packet residual error rate--number of lost or
corrupted messages compared to total messages in the sampling period transfer
failure probability--the fraction of the time when the throughput, transit
delay, or residual error were not those agreed upon at the **start** of the
connection connection release delay--time between when one node initiates a
release and the other node performs the release connection release failure
probability--fraction of release attempts which do not succeed
protection--specifies a secure connection
priority--indicates traffic priority
over the connection resilience--probability that the transport layer
spontaneously terminates

Detailed Description Text - DETX (841):
Transaction Services are typically used in three-tier and multi-tier architectures. Particularly in Netcentric environments, applications might need to support getting and providing **access to multiple back-end** services, across enterprises, as part of a single transaction or **user** activity. This can be especially challenging if the user does not own some or all of the back-end services and/or data that its application relies on.

Detailed Description Text - DETX (864):
Brief Product Considerations Encina--The Encina DTP (OLTP) was built on top of OSF's DCE. This is both its greatest asset and curse. DCE offers a very complete set of functions including security services, RPC's, a directory service (like a yellow pages for clients to find services) and a standard time service, and it is truly cross-platform and is endorsed by most vendors. The problem is that it is a resource hog, and is fairly slow. DCE is also somewhat immature in that there are not many tools to help you administer or program applications (although many are on the way). Encina adds primarily a transactional element and some load balancing services to RPC's. It also provides an easier interface to work with (although it is still an administrative nightmare). The good news is that the tools are getting better all of the time. Also, Encina is very scalable and services can be on any machine in the network. Finally, Encina's load balancing is quite good, much better then native DCE or Tuxedo. Tuxedo Functionality Can handle a large number of concurrent client applications Can handle a large volume of through-put (ex. 1000+TPS) Scaleable (handle many clients

or a few without
code rewrite) Supports Transactions, including XA
transactions Has its own
transaction resource manager Guaranteed message delivery
using a stable storage
queue (/Q) Future service delivery using/Q (usually for
batch processing) Can
prioritize messages--most important get processed sooner.
Supports many
platforms (all UNIX, NT, all common **client** platforms)
Tuxedo supports C, C++,
and Cobol development Can be used for basic c/s messaging
Supports
conversational messaging between a **client** and a specific
server Peer-to-peer,
client-to-**client** messaging is supported Unsolicited
messaging is supported for
client processes Asynchronous:service calls can be made by
client and server
processes Synchronous service calls can be made by **client**
and server processes
Synchronous calls that receive no return message are
supported Very good
security--must connect to access services Security can be
integrated w/Kerberos
Has many different buffer types: view to pass C structs,
FML to pass anything,
carrays to pass binary (sound, video), strings to pass
strings FML allows
dynamic messages to be sent/received Automatic error
logging for Tuxedo
components (ULOG, tagent log) Application code can write to
the ULOG with a
Tuxedo API (error logging provided) Automatic process
monitor for process that
die or machines that get partitioned Service location
independency
(distribution/directory services) Platform
independency-handles **data** conversion
Built in **data** compression (if desired) Built in performance
measurement feature
for services Built in admin functions to monitor Tuxedo
system online (tmadmin)
A server can be called based on **data** in the message (**Data**
Dependent Routing)
Customizable server **start-up** and shutdown functions are
automatically called.

/Domains allow independent Tuxedo regions to share services Extensions to
execute IMS and CICS transactions from UNIX (Open
Transport) Subscribe and
Broadcast supported APIs to get admin and system monitoring data for custom
operation tools JOLT (java to access Tuxedo servers) Other Reasons to Use
Tuxedo Tuxedo is the market leader OLTP Tuxedo is a proven product used in
mission critical systems govt. and financial) Tuxedo can be used to develop
highly-available systems (24.times.7) Has been implemented with PowerBuilder,
VisualBasic, Motif clients, and unix batch systems. Cons of Using Tuxedo
Tuxedo for basic c/s messaging is overkill. Expensive to purchase Can be
complicated to develop with and administer System performance tuning requires
an experienced Tuxedo administrator Uses IPC resources and therefore should not
be on same machine w/other IPC products Must be understood thoroughly before
design starts. If used incorrectly, can be very costly. Single threaded
servers requires an upfront packaging design. Difficult to debug servers Does
not work well with Pure Software products: Purify, Quantify Servers must be
programmed to support client context data management Difficult to do asynch
messaging in 3rd party Windows 3.x client tools (ex. PowerBuilder)

Detailed Description Text - DETX (882):
Runtime services convert non-compiled computer languages into machine code
during the **execution of a program**.

Detailed Description Text - DETX (886):
VBRUN300.DLL VBRUN300.DLL--**runtime** Dynamic Link Library that supports
programs written in Visual Basic.

Detailed Description Text - DETX (888):
Typically, a Virtual Machine is implemented in **software** on top of an operating system, and is used to **run** applications. The Virtual Machine provides a layer of abstraction between the applications and the underlying operating system and is often used to support operating system independence.

Detailed Description Text - DETX (891):
Virtual machines such as the Java virtual machine or the Smalltalk virtual machine implement their own versions of operating system services in order to provide the application with complete platform independence. Java virtual machine--**software** implementation of a "CPU" designed to **run** compiled Java byte code. This includes stand-alone Java applications as well as "applets" that are downloaded and run in Web browsers. Smalltalk virtual machine--runtime engine that interprets application code during execution and supports platform independence.

Detailed Description Text - DETX (900):
All MS Windows based application maintain their own profile file (XXXXXXXX.INI) that is used during application startup, execution, and shutdown. This is a flat text file that contains information that can be used by the application during various phases of execution. For example, if an application needs to connect to a **database** engine/server, it needs to know, during **startup, various information** like--**database** name, the server name, login ID, etc. Instead of hard coding all these **information** in the application executable, this **information** can be stored in the profile file for flexibility.

In the future, if the database server name should change, this change only
needs to be entered in the applications profile file.

Detailed Description Text - DETX (903):
Environment Verification Services ensure functionality by monitoring,
identifying and validating environment integrity prior and during **program execution**. (e.g., free disk space, monitor resolution, correct version).
These services are invoked when an application begins processing or when a
component is called. Applications can use these services to verify that the
correct versions of required Execution Architecture components and other
application components are available.

Detailed Description Text - DETX (906):
ActiveX framework provides services for automatic installation and upgrade
of ActiveX controls. When using IE, i.e., Microsoft's Web browser, because of
its integration with **Windows OS,** ActiveX controls can be automatically
installed and automatically upgraded on the users machine without the developer
adding any additional code.

Detailed Description Text - DETX (908):
Task & Memory Management Services allow applications and/or other events to
control individual computer tasks or processes, and manage memory. They
provide services for scheduling, **starting,** stopping, and restarting both **client**
and server tasks (e.g., software agents).

Detailed Description Text - DETX (979):
Web Server Services enable organizations to manage and publish information
and deploy Netcentric applications over the Internet and

intranet environments.
These services support the following: Managing documents in most formats such
as HTML, Microsoft Word, etc. Handling of client requests for HTML pages. A
Web browser initiates an HTTP request to the Web server either specifying the
HTML document to send back to the browser or the server **program** (e.g., CGI,
ASP) to **execute**. If the server **program** is specified, the Web server **executes**
the program which generally returns a formatted HTML page to the Web Server.
The Web server then passes this HTML page just as it would any standard HTML
document back to the Web browser. Processing scripts such as Common Gateway
Interface (CGI), Active Server Pages (ASP). Server side scripting enables
programs or commands to be **executed** on the server machine providing access to
resources stored both inside and outside of the Web server environment. For
example, server side scripts can be used to process requests for additional
information, such as data from an RDBMS. Caching Web pages. The first time a
user requests a Web page, the Web server retrieves that page from the network
and stores it temporarily in a cache (memory on the Web server). When another
page or the same page is requested, the Web server first checks to see if the
page is available in the cache. If the page is available, then the Web server
retrieves it from the cache, otherwise it retrieves it from the network.
Clearly, the Web server can retrieve the page from the cache more quickly than
retrieving the page again from its location out on the network. The Web server
typically provides an option to verify whether the page has been updated since
the time it was placed in the cache, and if it has to get the latest update.

Detailed Description Text - DETX (1126):
If the business logic is stored and **executed** on the client, **software** distribution options must be considered. Usually the most expensive option is to have a system administrator or the user physically install new applications and update existing applications on each client machine. Another option is to use a tool that performs automatic software distribution functions. However, this option usually requires the software distribution tool to be loaded first on each client machine. Another option is to package the application into ActiveX controls, utilizing the automatic install/update capabilities available with ActiveX controls--if the application is launched from a Web browser.

Detailed Description Text - DETX (1132):
If the business logic is stored and **executed** on the client, **software** distribution options must be considered. Usually the most expensive option is to have a system administrator or the user physically install new applications and update existing applications on each client machine. Another option is to use a tool that performs automatic software distribution functions. However, this option usually requires the software distribution tool to be loaded first on each client machine. Another option is to package the application into ActiveX controls, utilizing the automatic install/update capabilities available with ActiveX controls--if the application is launched from a Web browser.

Detailed Description Text - DETX (1234):
Physical components play a critical role in net-centric computing because they can be distributed, as encapsulated units of **executable software,**

throughout a heterogeneous environment such as the Internet. They have the ability to make the Web more than a toy for retrieving and downloading information. Robert Orfali, Dan Harkey, and Jeri Edwards, well-known experts in the field of component- and object-based development, wrote the following about distributed objects (same as "distributed components" for the purpose of this discussion):

Detailed Description Text - DETX (1319):
As with **client**/server, architecture work must **start** early. As noted above, this is particularly challenging because of the level of application reuse in a well-designed application framework and domain component model. Because of this reuse, the framework must be heavily driven by application requirements, or scenarios. Yet, the architecture team must stay one step ahead of application development teams to ensure that the architecture and component model are ready in time to be reused. Thus, a difficult tension exists between scenarios and frameworks.

Detailed Description Text - DETX (1333):
There is still not enough experience with component technology to support rigorous, detailed metrics. One reasonable checkpoint for estimating an initial project is to use traditional techniques, and then add time to adjust for contingency and start-up costs such as training, learning curve, and architecture development. Early **client** engagements have demonstrated that an initial project may almost always be more expensive due to these **start-up** costs.

Detailed Description Text - DETX (1456):
Architecture roles must be defined to support this greater degree of specialization. One engagement used the following partitioning strategy: Functional architect-responsible for resolving decisions for what the system should do. This person is ideally a user with a solid understanding of systems, or a systems person with a good understanding of, and relationship with, the users. Technology architect-responsible for delivering the platform, systems **software,** and middleware infrastructure to support **execution,** development, and operations architectures. User interface architect-responsible for setting direction of the user interface metaphor, layout standards, and integrated performance support (IPS). Application frameworks architect-responsible for designing, delivering, and supporting the application framework that provides the overall structure, or template, of the application. Object model architect-responsible for identifying and resolving modeling issues necessary to achieve a high degree of business reuse and modeling consistency.

Detailed Description Text - DETX (1537):
The need to **start** architecture implementation early is well-understood for traditional or component-based **client**/server development. What is different with component-based development, however, is the need for the component model to lead the application and user interface development.

Detailed Description Text - DETX (1543):
As with **client**/server, architecture work must **start** early. As noted above, this is particularly challenging because of the level of application reuse in a well-designed application framework and domain component

model. Because of
this reuse, the framework must be heavily driven by application requirements,
or use cases. Yet, the architecture team must stay one step ahead of
application development teams to ensure that the architecture and component
model are ready in time to be reused. Thus, a difficult tension exists between
use cases and frameworks.

Detailed Description Text - DETX (1749):
To maximize reuse requires the assembly and configuration of **run**-time
components in addition to being able to construct new components as part of the
software construction process. A new breed of tools supporting black box reuse
referred to as "Component managers" should be considered one of the primary
tools provided with the environment to 1) support transformations between tools
where this may continued to be a requirement, 2) enable component views of
reuse allowing configuration from both run-time and development components and
3) give component developers security features preventing users from modifying
and/or reusing certain components if they desire. It requires the ability to
categorize components and search components according to property descriptions
in a way that can be ascertained without the viewing of source code.

Detailed Description Text - DETX (1756):
ODM has many predefined deliverable templates that are targeted towards this
suite of tools including Word, Excel and Visio templates. Often times
management under-estimates the **start** up cost of integrating the tools in such a
way as to improve the flow of **information** between phases and for ensuring that
information is published to the team in a way that is

accessible and plentiful.
However project experience teaches that this investment can yield many returns
down the road if the development architecture includes processes and
infrastructure to support this flow of information.

Detailed Description Text - DETX (1800):
The latter approach provides a leverage point for performance tuning the
initialization of the window. The processing may be tuned to use a more
efficient algorithm; the results of the **initialization** may be cached during
application packaging and read during **initialization**; or, efficient
initialization methods may be generated and maintained automatically from the
information by a code generator once it becomes clear what the most efficient
implementation is. In any case, the flexibility provided by this leverage
point allows many more possibilities to be considered during performance
tuning. Note that all three optimizations could be achieved without manually
visiting a single of perhaps hundreds of windows which share the initialization
processing.

Detailed Description Text - DETX (1824):
In general, batch still has the following fundamental characteristics:
Scheduling--Services are required to manage the flow of processing within and
between batch jobs, the interdependencies of applications and resources as well
as to provide integration with checkpointing facilities.
Restart/Recovery--Batch jobs must be designed with restartability in mind.
This implies the need for a batch restart/recovery architecture used to
automatically recover and re-start batch **programs** if they should fail during
execution. Controls--Run-to-run and integrity controls are

still required to
ensure that all data is processed completely.
Reporting--Services are required
to handle configurable report creation, distribution,
printing and archiving.

Detailed Description Text - DETX (1851):
where "abstractType" is the type of the common abstract interface, and key
is a piece of information which identifies the appropriate concrete type.
(This could be the same piece of information used in the switch/case statement;
there could be a variety of ways to get it). When this method is invoked, the
Abstraction Factory consults its internal mapping and creates an "empty" object
of the proper concrete class. The factory then casts the concrete object into
the abstraction and returns it to the method's client. This **client** (a
framework most likely) will then instruct the abstraction to **initialize** itself
from the incoming **data** stream.

Detailed Description Text - DETX (1862):
Most business systems today include some sort of batch processing. Batch
processing is the **execution** of a series of **instructions** that do not require any
interaction with a user to complete. Batch jobs are usually stored up during
the day and executed during evening hours when the system load is typically
lower.

Detailed Description Text - DETX (1868):
Therefore, represent each type of batch job in the system as its own class.
An abstract class (BatchJob) will exist from which all specific types of batch
jobs will derive from. The abstract BatchJob contains **data** that all batch jobs
require: name, current status (pending, **started,** finished,

deleted), messages
encountered during its run, various times (submission, **start,** completion), and
a priority, for example. It also should provide some default behaviors
including running the job and logic to execute before and after the run.

Detailed Description Text - DETX (2048):
As an additional option for this aspect, the legacy wrapper component may
also include a pure legacy wrapper component. As even a further option to this
aspect, the legacy wrapper component may include a hybrid legacy wrapper
component. Also, the interfacing may further include: sending a message from
the client to the legacy wrapper component via the component integration
architecture; sending the message via the component adapter to the legacy
integration architecture; forwarding the message to the legacy adapter;
formatting the message to match an application **program** interface (API) of the
legacy system; **executing** calls on the legacy system based on the formatted
message; **executing** function of the calls and returning results to the legacy
adapter, legacy integration architecture, component adapter, and legacy wrapper
component which reformats the results; and forwarding the reformatted results
to the client via the component integration architecture.

Detailed Description Text - DETX (2050):
A legacy system is an existing system that does not conform to the
technology, architecture and standards of the current project. A large IBM
3090 Mainframe **running programs** to calculate automobile insurance rates is an
example of a Legacy System. It is large, very important to an insurance
company, and runs on older, proprietary hardware.

Detailed Description Text - DETX (2156):
Definitions Starting Key The Starting Key is the initial starting point for the search. The **database** will begin searching for **data** (customers in the message trace above) at the **Starting** Key. An example starting key could be "A*". Last Found Key The Last Found Key is used to request subsequent pages of data from the Server and the database. The "last found key" defines the **starting** point for the next **data** request. The Server will begin searching for data at the "last found key" and continue until it has retrieved a full "page" of information. When all of the data has been retrieved from the Server and Database, the Last Found Key is left blank. This notifies the Client that all the data has been sent. Intermediate Page An intermediate "page" is returned for every request but the last. When a client receives an intermediate page and a "last found key", the client knows more "pages" of data exist on the server. In order to obtain an intermediate "page," a "last found key" must be passed from the client to the server. When the Server has retrieved a full "page" of data, the new "last found key" is saved. It is then passed back with the intermediate "page." The new "last found key" defines the **starting** point for the next **data** request. Last Page When the Server has retrieved all of the data meeting the search criteria, the Server builds the last "page." When the last page is returned to the client, the "last found key" is left blank. This notifies the client the search is complete and no more data matching the search exists on the Server. Note that the last page is usually smaller than the other pages. Empty Page When no data are selected from the search criteria,

the server builds an empty page signaling to the client no more data exist on
the server. Static or Dynamic Page size The page size can be defined
statically or dynamically. The message trace diagram in FIG. 99 depicts a
static page size. If you'd like a dynamic page size, the client must pass an
additional parameter with each request to the Server. The additional parameter
would be the page size.

Detailed Description Text - DETX (2158):
Collaborations 1. The user "clicks" the "Get Customers" button on the User
Interface. The **Client** UI makes a getAllCustomers request of the Server and
passes a **Starting** Key as a parameter. Since the user wants to view all of the
customers, a Starting Key of spaces is used. Message sent=getAllCustomers(" ")
2. The Server receives the request from the Client. The Server realizes the
Starting Key is blank and knows this is a new request. Thus, the Server
requests first four customers (the page size) from the database. 3. The
database returns the first four customers (Albert Abraham, Ned Abraham, Sally
Abraham and Alice Allen) and a "Last Found Key" ("Alice Allen") to the Server.
The "Last Found Key" denotes the last entry found during the search. It will
be used for subsequent searches. 4. The Server builds a page with the four
customers retrieved from the database. The Server returns the page and the
Last Found Key to the Client. Page Type=Intermediate Page="Albert Abraham",
"Ned Abraham", "Sally Abraham" & "Alice Allen"
lastFoundKey="Alice Allen"

Detailed Description Text - DETX (2697):
Therefore, detach the state of the business object into a separate state

class that can be agreed upon and completed prior to the **start** of significant
development by either the **data** access team or the business object team. This
class should be nothing more than the raw data (preferably basic data types)
used to represent the state of the object. The business object contains all
business logic and a reference/pointer to an instance of the respective state
class. This allows the development of business logic and data access to
proceed in parallel with the state class serving as a contract between the two
teams.

Detailed Description Text - DETX (2737):
While a transaction is in process, the state of the business model may not
be consistent, so it is necessary to manage the entire transaction from its
point of origin to its point of completion. Whether the transaction is
successful or not, the point of completion will always result in a steady,
consistent state for the business model. For successful transactions, data
changes will be committed and the business model will reflect all new business
data associated with the transaction. For failed transactions, **data** changes
will be rolled back and the business model will appear as it did prior to the
start of the transaction.

Detailed Description Paragraph Table - DETL (20):
... DATA DIVISION. FD FILE-STREAM-IN RECORD CONTAINS 100 CHARACTERS ...
WORKING-STORAGE SECTION. 01 WS-FILE-STREAM-IN PIC X(100). 01
WS-SHARED-FORMAT-HEARDER 03 WS-HEADER-OBJECT-TYPE PIC X(10). 03
WS-HEADER-NUM-OF-ATTRIBUTES PIC X(7). 03
WS-HEADER-VERSION-OF-FORMAT PIC 999.
01 WS-SHARED-FORMAT-ATTRIBUTE 03 WS-ATTRIBUTE-NAME PIC

```
X(5).  03
WS-ATTRIBUTE-TYPE PIC X(5).  03 WS-ATTRIBUTE-SIZE PIC 999.
01 TEMP-VARIABLES
03 WS-INDEX PIC 9999.  ... 01 WS-CUSTOMER  03 WS-NAME PIC
X(10).  03 WS-SEX PIC
X(7).  03 WS-AGE PIC 999.  ... 88 LT-HEADER-SIZE PIX 99
VALUE 20.  88
LT-ATTRIBUTE-DESCRIPTOR-SIZE PIX X(1) VALUE 14.  88
LT-DELIMINATOR PIX X(1)
VALUE ".vertline.".  88 LT-STRING PIX X(1) VALUE "STG ".
88 LT-NUMBER PIX X(1)
VALUE "NUM ".  ... PROCEDURE DIVISION.  ... *** OPEN THE
FILE STREAM AND READ
IT INTO THE  TEMPORARY ***  *** VARIABLE WS-FILE-STREAM-IN
***  OPEN
FILE-STREAM-IN.  READ FILE-STREAM-IN INTO WS-FILE-STREAM-IN
 AT-END CLOSE
FILE-STREAM-IN  END-READ.  *** MOVE THE HEADER INFORMATION
INTO THE HEADER
COPYBOOK****  MOVE (WS-FILE-STREAM-IN FROM ZERO TO
LT-HEADER-SIZE)  TO
WS-SHARED-FORMAT-HEADER.  *** FIND WHAT BYTE THE DATA
STARTS AT AND SET THE
INDEX ****  MOVE (LT-ATTRIBUTE-DESCRIPTOR-SIZE *
WS-HEADER-NUM-  OF-ATTRIBTES)
TO WS-INDEX.  *** PARSE THE APPROPRIATE OBJECT STRUCTURE
OFF OF ***  *** THE
STREAM ***  IF WS-HEADER-OBJECT-TYPE EQUALS "CUSTOMER "
THEN  PERFORM
1000-PARSE-CUSTOMER-STREAM THRU
1000-PARSE-CUSTOMER-STREAM-END.  ELSE IF
WS-HEADER-OBJECT-TYPE EQUALS "EMPLOYEE "  THEN  ... ELSE IF
 ... ELSE  *** END
THE PROGRAM  RUN-STOP.  END-IF.
1000-PARSE-CUSTOMER-STREAM.  *** READ WHICH
VARIABLE IT IS AND POPULATE THE  CORRECT ***  *** VARIABLES
***  IF
(WS-FILE-STREAM FROM WS-INDEX TO (WS-INDEX +5)) =  "NAME "
THEN  MOVE WS-INDEX
TO START-INDEX.  *** FIND THE DELIMINATOR AFTER THE NAME
STRING AND  ***  ***
MOVE THE NAME VALUE INTO THE SEX VARIABLE ****  PERFORM
VARYING WS-INDEX  FROM
START-INDEX  BY 1  UNTIL (WS-FILE-STREAM-IN AT INDEX) = LT-
 DELIMINATOR
END-PERFORM.  MOVE (WS-FILE-STREAM FROM START-INDEX  TO
WS-INDEX) TO WS-SEX.
```

PERFORM 1000-PARSE-CUSTOMER-STREAM THRU
1OOO-PARSE-CUSTOMER-STREAM-END. ELSE
IF (WS-FILE-STREAM FROM WS-INDEX TO (WS-INDEX + 5)) = "SEX
" THEN *** FIND
THE DELIMINATOR AFTER THE SEX STRING AND MOVE *** *** THE
SEX VALUE INTO THE
SEX VARIABLE *** MOVE WS-INDEX TO START-INDEX. PERFORM
VARYING WS-INDEX
FROM START-INDEX BY 1 UNTIL (WS-FILE-STREAM-IN AT
WS-INDEX) = LT-
DELIMINATOR END-PERFORM MOVE (WS-FILE-STREAM FROM
START-INDEX TO WS-INDEX)TO
WS-SEX PERFORM 1000-PARSE-CUSTOMER-STREAM THRU
1OOO-PARSE-CUSTOMER-STREAM-END. ELSE IF (WS-FILE-STREAM
FROM WS-INDEX TO
(WS-INDEX +5)) = "AGE " THEN *** FIND THE DELIMINATOR
AFTER THE AGE STRING
AND *** *** MOVE THE AGE VALUE INTO THE AGE VARIABLE ****
MOVE INDEX TO
START-INDEX. PERFORM VARYING WS-INDEX FROM START-INDEX
BY 1 UNTIL
(WS-FILE-STREAM-IN AT WS-INDEX) = LT- DELIMINATOR
END-PERFORM MOVE
(WS-FILE-STREAM FROM START-INDEX TO WS-INDEX)TO WS-AGE
PERFORM
1000-PARSE-CUSTOMER-STREAM THRU
1OOO-PARSE-CUTOMER-STREAM-END. ELSE PERFORM
2000-SAVE-CUSTOMER THRU 2000-SAVE-CUSTOMER-END. END-IF.
1000-PARSE-CUSTOMER-STREAM-EXIT. 2000-SAVE-CUSTOMER. ***
CALL A SQL MODULE TO
SAVE THIS INFORMATION IN THE *** RELATIONAL DATABASE
CALL
"SAVE-CUSTOMER-IN-DATABASE" USING WS- CUSTOMER ...
2000-SAVE-CUSTOMER-END.
Conversely, a stream could be created by a non-object
system or another object
system, populated with customer information, and sent to
one's object-based
system. Once in the object-based system, the CustomerObject
could use a
"streamOff: aSteam" method, instanciate a CustomerObject,
and populate it with
the appropriate attribute values.

Claims Text - CLTX (1):

1. A method for distributing incoming requests from a

client amongst server components for optimizing usage of resources, comprising the steps of: (a) receiving incoming requests from a user interface, wherein the user interface resides on a client and the requests are received by an activity module, and wherein the activity module instructs the client to handle a first subset of the requests on the client and the activity module forwards a second subset of the requests to a utilization-based load balancer, such that the activity module resides between the user interface and a plurality of server components; (b) storing the second subset of the requests on the load balancer upon receipt thereof from the activity module; (c) determining an availability of server components from among the plurality of server components; (d) compiling a listing of available server components; (e) determining which server component on the listing of available server components is most appropriate to receive each of the second subset of the requests, wherein the **load** balancer calculates an amount of utilization that each available server component is currently experiencing wherein the amount of utilization of each available server components is calculated based on at least two of: current CPU utilization, **kernel** scheduling run-queue length, current network traffic at a node to the server component, and a number of requests currently being serviced; and (f) sending each of the second subset of the requests to the selected server component determined to be most appropriate to receive each of the second subset of the requests.

Claims Text - CLTX (5):

5. A computer program embodied on a computer readable medium for

distributing incoming requests from a client amongst server components for optimizing usage of resources, comprising the steps of: (a) a code segment that receives incoming requests from a user interface, wherein the user interface resides on a client and the requests are received by an activity module, and wherein the activity module instructs the client to handle a first subset of the requests on the client and the activity module forwards a second subset of the requests to a utilization-based load balancer, such that the activity module resides between the user interface and a plurality of server components; (b) a code segment that stores the second subset of the requests on the load balancer upon receipt thereof from the activity module; (c) a code segment that determines an availability of server components from among the plurality of server components; (d) a code segment that compiles a listing of available server components; (e) a code segment that determines which server component on the listing of available server components is most appropriate to receive each of the second subset of the requests, wherein the **load** balancer calculates an amount of utilization that each available server component is currently experiencing wherein the amount of utilization of each available server components is calculated based on at least two of: current CPU utilization, **kernel** scheduling run-queue length, current network traffic at a node to the server component, and a number of requests currently being serviced; and (f) a code segment that sends each of the second subset of the requests to the selected server component determined to be most appropriate to receive each of the second subset of the requests.

Claims Text - CLTX (9):

9. A system for distributing incoming requests from a client amongst server components for optimizing usage of resources, comprising the steps of: (a) logic that receives incoming requests from a user interface, wherein the user interface resides on a client and the requests are received by an activity module, and wherein the activity module instructs the client to handle a first subset of the requests on the client and the activity module forwards a second subset of the requests to a utilization-based load balancer, such that the activity module resides between the user interface and a plurality of server components; (b) logic that stores the second subset of the requests on the load balancer upon receipt thereof from the activity module; (c) logic that determines an availability of server components from among the plurality of server components; (d) logic that compiles a listing of available server components; (e) logic that determines which server component on the listing of available server components is most appropriate to receive each of the second subset of the requests, wherein the **load** balancer calculates an amount of utilization that each available server component is currently experiencing wherein the amount of utilization of each available server components is calculated based on at least two of: current CPU utilization, **kernel** scheduling run-queue length, current network traffic at a node to the server component, and a number of requests currently being serviced; and (f) logic that sends each of the second subset of the requests to the selected server component determined to be most appropriate to receive each of the second subset of the requests.